FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Otto Poikajärvi

CONCURRENT IMAGE CAPTURE IN A RASPBERRY PI NETWORK

Bachelor's Thesis
Degree Programme in Computer Science and Engineering
November 2019

Poikajärvi O. (2019) Concurrent Image Capture in a Raspberry Pi Network. University of Oulu, Degree Programme in Computer Science and Engineering, 25 p.

ABSTRACT

3D scanners are used to create digital 3D models from real life objects. Some 3D scanners work by moving a single camera around the scan target object, others by having many cameras around the object. Multi camera 3D scanners are useful for scanning objects that can not stay still for extended periods, but they require synchronized image capture for the different cameras. Raspberry Pi is a small computer that can perform the necessary tasks to function as a camera unit in a multi camera 3D scanner.

In this work an implementation that allows for concurrent image capture for a self-configuring Raspberry Pi network is presented. The goal is to create a system that allows a user to use concurrent image capture without having to rely on a central controlling device. The implementation uses multicast sockets for synchronizing image capture and communication between different Raspberry Pi units. Evaluation is performed for the image capture synchronization and communication protocol.

Evaluation results indicate that the solution developed could be used as a basis for future 3D scanner development but its functionality as a 3D scanner is not tested. Current version lacks certain features such as image deleting.

Keywords: service discovery, multicast, 3D scanning, 3D scanner

Poikajärvi O. (2019) Yhtäaikainen kuvien otto Raspberry Pi-verkostossa. Oulun yliopisto, Tietotekniikan tutkinto-ohjelma, 25 s.

TIIVISTELMÄ

3D-skannereita käytetään digitaalisten 3D-mallien luomiseen oikean maailman esineistä. Jotkin 3D-skannerit toimivat liikuttamalla yksittäistä kameraa skannattavan kohteen ympärillä, toiset käyttävät sen sijaan useampaa kameraa kohteen ympärillä. Useampaa kameraa käyttävät 3D-skannerit ovat hyödyllisiä kohteille, jotka eivät voi pysyä paikallaan kauaa, mutta ne myös tarvitsevat synkronoitua kuvien ottamista. Raspberry Pi on pieni tietokone, joka kykenee 3D-skannerin kameralta vaadittuihin tehtäviin.

Tässä työssä esitellään implementaatio, joka mahdollistaa yhtäaikaisen kuvien oton itsensä konfiguroivassa Raspberry Pi-verkostossa. Projektin tavoite on luoda järjestelmä, joka mahdollistaa käyttäjän käyttää samanaikaista kuvien ottoa ilman riippuvuutta jostakin keskusyksiköstä. Implementaatio käyttää multicast-pistokkeita kuvien synkronoituun ottamiseen ja Raspberry Pi:den väliseen kommunikaatioon. Evaluaatio suoritetaan kuvien ottamisen synkronaatiolle ja kommunikaatioprotokollalle.

Evaluaation tulokset viittaavat siihen, että implementaatiota voisi käyttää 3D-skanneri kehityksen pohjana. Implementaation toimintaa 3D-skannerina ei testata. Nykyisestä versiosta puuttuu joitain ominaisuuksia, kuten kuvien poisto.

Avainsanat: palvelun etsintä, multicast, 3D-skannaus, 3D-skanneri

TABLE OF CONTENTS

1. INTRODUCTION

3D scanners are used to create 3D models of real life objects. 3D scanners are usually either single camera scanners that rotate either the scanned object or the camera around the object or multi camera scanners that have cameras surrounding the scanned object. Specialized software is required to use the different pictures captured by scanners to create 3D models.

Straub et al. [1] divide 3D scanning to three different size scales. Objects significantly larger than humans, human-sized objects and objects significantly smaller than humans require different scanning procedures and have different uses. Objects significantly larger than humans include buildings and entire scenes. These scans can be used e.g. when designing urban environments, creating virtual presentations of real estates or historical sites, or doing mining assessment. For human-sized scanning they mention use cases such as assessing the shape of someone's back for medical purposes and clothes tailoring and that scanning objects significantly smaller than humans is useful when e.g. measuring facial expressions, reverse engineering or creating object databases for historical artifacts or product development. Virtual environments such as video games or virtual reality applications can utilize different 3D scanning scales at the same. Virtual museum could for example utilize the user's self-scan as a basis for an avatar. It could then show the user 3D scanned historical artifacts at their original size while navigating a model of some historically important building as that avatar.

Raspberry Pi is a credit card sized computer that can perform tasks needed in many different kinds of professional and hobbyist projects. Raspberry Pi has support for Ethernet and wireless local area network connectivity, it has wide support for tertiary devices including cameras and it can run a server for a website. Features such as these have been used when building wireless sensor networks, smart home implementations and 3D scanners. However 3D scanners based on the Raspberry Pi often do not take full advantage of the hardware and instead rely on using Windows computers as controlling master devices.

1.1. The Project Goal

This project's goal is an implementation that provides concurrent image capture for Raspberry Pis, but is not reliant on an external master device. Advantages of such a system compared to existing multi camera 3D scanners utilizing Raspberry Pis are flexibility in how the scanner's interface is provided for the user and being less reliant on a separate computer with special software installed. A system like that could also be repurposed for other multi camera uses, such as panorama image capture. Independent 3D scanner implementation requires combining networking features such as service discovery with the 3D scanner's camera units. An interface that provides the user options to capture and view images has to be available. Each Raspberry Pi unit has to work independently and together with other units, each providing their own user interface through a website.

This work utilizes service discovery to allow the Raspberry Pis to know each other's addresses within their network. Communication between the Raspberry Pis happens through User Datagram Protocol (UDP) using multicast to send messages to all devices

in the network at once. Each Raspberry Pi keeps a list of other devices in the network that use the service discovery implementation. This list is then used for variety of purposes, from fetching images from different Raspberry Pis to displaying how many devices are available to the user.

Concurrent image capture is also achieved with multicast. Each Raspberry Pi receives the command to capture an image at roughly the same time. A Raspberry Pi can then download the images from other Raspberry Pis when the user requests it. Raspberry Pi's standard program called raspistill is used for capturing the images. Many already existing 3D scanners, such as Pi3DScan's [2] and Straub and Kerlin's [3] scanners, use multicast to achieve concurrent image capture but unlike their versions, this work also studies its synchronization in wireless environments.

The user interface is accessible through a website provided by each Raspberry Pi. The user interface includes option for the user to start image capture, view images saved on that Raspberry Pi, view live footage captures by that Raspberry Pi's camera and request the Raspberry Pi to download all images belonging to a single set from different Raspberry Pis in the network. The user interface is tested to function on both PCs and mobile devices.

Utilizing the implementation as a 3D scanner requires placing many Raspberry Pis around the object being scanned and adjusting the environment around the object to ensure that the model can be created properly. Different solutions for 3D model creation software require different things from the environment, for example reference points. This aspect of the implementation is however not tested because of hardware limitations.

1.2. Contents of the Thesis

This work is split to six chapters, with this intro chapter explaining the project and its goals. Chapter two presents how Raspberry Pi or similar small power devices are used in Internet of Things (IoT) solutions and 3D scanners. Chapter three documents the technical implementation of the project. Implementation of multicast, user interface and how different parts of the implementation fit together is detailed. Chapter four includes testing of synchronization of image capture and service discovery device limits and response time. Results of the tests that are then discussed in chapter five. The implementation's features are compared to already existing 3D scanners in chapter five also. Chapter six concludes and summarizes the thesis.

This work presents an implementation that allows concurrent image capture with multiple Raspberry Pis that configure themselves to work as a singular unit. The user interface created is usable with both computers and mobile devices. Service discovery implementation's support for large amount of devices is evaluated. Implementation's use as a 3D scanner is not tested.

2. RELATED WORK

2.1. Raspberry Pi as a Device

Raspberry foundation presents [4] Raspberry Pi as a small computer useful for both desktop use and for coding and electronics projects. Raspberry Pi's operating system is an ARM Linux distribution, Raspberry Pi foundation recommends Raspbian.

Vujović and Maksimović [5] compare first generation Raspberry Pi's performances and constraints with other available wireless sensor network (WSN) platforms. According to them Raspberry Pi's advantages are large working memory, expandable data storage, amount of processing power, USB 2.0 peripheral support, possible expansions and adapters and Linux support. Disadvantages include lack of real-time clock, mandatory booting from an SD card even if USB booting would be preferred, and power consumption. They also note that Raspberry Pi running a full Linux operating system makes programming and thus solution implementation simple when compared to micro controllers which often rely on development kits. It is also very difficult to use a microcontroller for multitasking [6]. It should be noted that according to Raspberry Pi Foundation's own measurements later versions of Raspberry Pi use even more power, average power draw being sometimes over 3 times higher with Raspberry Pi 4B than with Raspberry Pi 1B+ [4].

Lewis, Campbell and Stavroulakis [7] present similar claims about Raspberry Pi's advantages over lower power devices to Vujović and Maksimović [5]. Compared to something like Arduino, Raspberry Pi has more processing power and memory, and thus can provide a web server interface and operate as a data server [7].

2.2. Wireless Sensor Networks and Service Discovery

Raspberry Pis have to communicate with each other to function as a single unit. There are several approaches to how this could be done. Vujović and Maksimović [8] list three types of home automation systems, individual control devices, distributed-control systems and centrally controlled systems. Individual control devices have one appliance or function controlled by one device, for example a small device that turns a light switch on or off. Distributed-control systems allow appliances to communicate internally meaning the decision to turn a light switch on or off could be based on communication from different sensors in the home automation system. Centrally controlled systems have a central computer that routes the communications between the appliances. System of multiple Raspberry Pis connected to each other for camera purposes is not directly comparable with home automation systems but it can be seen as distributed-control system or a centrally controlled system depending on how the user connects to the system and how user's commands are distributed to different units. Central computer is a single point of failure in centrally controlled systems [8] and considering that all Raspberry Pis have the same software, can perform the same tasks and the system would preferably be scalable to dozens of Raspberry Pis, there is no real need to rely on a single central computer.

Several IoT devices can function as a single unit for example in WSNs. In order to function as a single unit the devices have to be aware of each other. This is achieved

with service discovery. In 2015 Al-Fuqaha et al. [9] did a thorough look into the state of IoT, including WSNs and service discovery. Their WSN demonstration was based on three different types of motes/nodes, normal motes that just send packets to anchors that are closest to the sink, anchor motes which are special motes near normal motes and the sink mote which gathers the information from the other motes. Service discovery demo has every node broadcast packets detailing the node's service description. Other nodes then register the information received to their lookup tables, where the node can check which node to contact for a specific service. Source codes for the demos are available through GitHub.

Munir et al. [10] discuss the classifications of service discovery in wireless networks. They classify different solutions to two different types, centralized and decentralized. Centralized solutions have all service requests and their responses routed through directory agents, which are responsible for keeping a database for the available services. Decentralized or distributed solutions have no directory agents, instead user agents requesting a service multicast their request directly to service agents, which then respond with unicast. The discovery itself can use push or pull -model. In push the directory agents or servers advertise themselves, in pull the clients requesting the service ask for services. They position that push is better suited for commonly used services and pull for less often used services. Service discovery demo described by Al-Fuqaha et al. [9] would according to these classifications be a decentralized push based service discovery.

According to Quevedo et al. [11] service discovery in traditional networks is a well-studied subject, but in IoT networks issues are different, often requiring a high amount of automation for effective and efficient solutions. They note that service registries can be centralized for ease of management, but centralized systems are less scalable when adding new IoT devices to the network and that in many IoT applications security and privacy have to be considered when creating discovery solutions.

Cirani et al. [12] propose a service discovery mechanism for IoT networks that has IoT nodes that provide services, IoT Gateway that hosts description of resources available on other servers and manages the network, and client nodes that request the services of the IoT nodes. The mechanism assumes IP multicast can be used and that Dynamic Host Configuration Protocol is used for IP layer configuration. New nodes joining the network can either announce themselves to the IoT Gateway leading to IoT Gateway querying the node for its services and then adding it to the service list. Alternatively the IoT Gateway can periodically advertise itself in the network leading to the node answering and being added to the service list. Clients need to ask about available services from the IoT Gateway in order to know what services they can request.

Sruthy and George [13] describe a home surveillance system where a Raspberry Pi works as a master node as a part of a larger system. The user directly only connects to the master node and the node communicates with other devices that form the whole system. They note that small, low power and low cost computers like Raspberry Pi can be used to replace traditional computers in WSNs and that traditional computers in WSNs can be very expensive and consume a lot of power. This can be seen as a centrally controlled system, where the master node is the central computer. Master node collects information from other nodes and can be seen as a sink mote as described

in previous paper [9], though a master node does not simply collect data, but also controls the other nodes.

Vujović and Maksimović [8] detail a Raspberry Pi based WSN node, focusing on Apache Tomcat 7 based server solution. They use a REST architectural model, where clients and servers are stateless and the clients use methods to read data from the sensors of the WSN. They run the server with Apache Tomcat 7, because it is simple to configure and utilizes Raspberry Pis hardware well.

2.3. Different Smart Home Implementations

Smart homes use smart devices to automate processes, maximizing security, convenience, comfort, safety and energy-savings [8]. These smart devices range from simple sensors that translate physical events to information to complex internet connected systems.

Raspberry Pi has been used in many smart home related implementations similar to 2 previous publications [8, 13]. In many of these researches the Raspberry Pi both controls the sensors or sensor equivalents and runs the server which provides the collected information to users. Communication between Raspberry Pi and other devices in the implementations is facilitated with diverse solutions. Stojanoski et al. [14] set up Raspberry Pi as a server for their smart home implementation. Raspberry Pi connects to different smart devices within the household, both ON/OFF devices with just a simple switch and smart calling devices which have sensors and can transmit data. Raspberry Pi is set up after start up with an automatically running BASH script, that starts necessary listening scripts, defines variables and activates required pins. They use a Python listen script for taking the pictures due to performance optimizations. The listen script is a loop waiting for a command to take a picture, which it then stores in the storage of the device.

Patchava, Kandala and Babu [6] prototype a Raspberry Pi -based smart home automation technique. Their algorithm checks the ON/OFF status of appliances connected to Raspberry Pi and then displays it to the user through a website along with a video feed of the Raspberry Pi camera. The video feed is sent using MJPG-streamer, which is a simple and fast video streamer making it ideal for their home automation technique. Raspberry Pi is the central point of the whole system.

Rasyid, Saputra and Prasetiyo [15] design a portable smart home solution, where Raspberry Pi 3 is the main controller controlling Arduino-nodes. User of the smart home solution controls the nodes through a web interface that requires no additional applications from the user and works on a smart phone. Their server application is written using PHP with laravel framework, database uses MySQL and communication API is REST API with JSON for data exchange. The server runs on the Raspberry Pi.

Quadri and Sathish's [16] Raspberry Pi -based home automation and surveillance system is based around an Apache HTTP server running on Raspberry Pi. User controls the system's camera and door-mechanism through a website built using HTML and CSS. Python is used for scripts that initialize the system.

Lewis, Campbell and Stavroulakis [7] use a Python script for client-server communications in their digital environmental monitor. Their script runs a TCP/IP

data server, it sets up a socket for communication and then waits to be interrogated by a client. It responds to the interrogation with an array of sensor data.

Many of these smart home implementations use REST or similar designs [8, 15, 7] where the user can retrieve wanted information from the Raspberry Pi server with simple queries. Some [6, 16] solutions instead use website's features to control appliances connected to Raspberry Pi. Website based solutions presented are not as limited as REST-solutions when it comes to visualization and can show video and pictures to users. All these solution can be seen as centrally controlled systems as presented by Vujović and Maksimović [8].

Similarly to other website based designs Nguyen et al. [17] use Raspberry Pi in a low cost motion sensing camera to stream video data to a cloud server, where users can watch the video using a web browser. A Python script takes care of the streaming and being web browser based ensures support for mobile devices, such as iPhones and Android devices. Their web interface gives the users ability to save stills and full resolution video, capture a picture and change the resolution of the camera among other features.

2.4. 3D Scanners

According to Pi3DScan [2] 3D scanning can be used to make 3D models of real life objects, which can then be used for 3D printing, measurements or placing these models to games. They see Raspberry Pi's advantage when building 3D scanners being cheap price when comparing to older multicamera systems.

Straub and Kerlin [3] present a low cost instant 3D scanner with 50 Raspberry Pis. In their scanner Raspberry Pis are connected to each other with an Ethernet switch, which is also connected to a file server and a workstation. Raspberry Pis take pictures when they receive a multicast "capture now" message from the file server. If the user wants several scans taken, the file server generates the multicast message as many times as needed. Each Raspberry Pi receives the capture message nearly concurrently. Raspberry Pis first save the pictures on their SD cards before the pictures are sent to the file server. Using multicast ensures concurrent image capture, which is, according to Pi3DScan [2], necessary for 3D scanners when scanning something that can not stand still. Straub and Kerlin were heavily inspired by Pi3DScan's 3D scanner design [3].

Pi3DScan's [2] 3D scanner uses up to 100 Raspberry Pis as cameras, placed around the subject. Most Raspberry Pis work as camera units, some control lighting and projectors instead. All Raspberry Pis have the same listen script running, written in Python. Multi-cast messages are used to control all units simultaneously. Image retrieval from units and creation of multicast messages is done by Pi3DScan's proprietary software in their commercial solution, but open source version is also available. Commercial solution takes two sets of images, one with projectors off, another with them on. Use of projectors allows the user to scan items with little texture [2]. Machidon, Mihai and George's [18] FPGA-based scanner also takes two images per position, one with laser on, another with laser off before rotating the camera a certain amount around the object being scanned, though in their implementation the image without laser is used to find where the laser is in the other image.

There are open source 3D scanners that utilize only one Raspberry Pi. FreeLSS [19] is an open source 3D scanner platform using Raspberry Pi. FreeLSS uses only one Raspberry Pi with a turntable to perform the scan. It uses Raspicam library for video camera access. The scanner is used through a web interface running on the Raspberry Pi. Libmicrohttpd is used to run the interface server. Different pages on the website provide the user with different functions, the main interface is used when starting the scan or viewing previously taken pictures, camera page is used to check the camera feed and settings page gives user access to numerous changeable settings and the option to shutdown the system. Another similar single camera 3D scanner is FabScanPi [20]. FabScanPi's interface is browser based and it uses an Arduino controlled turntable for 360 degree view of the subject. Unlike multi-camera approaches, single camera solutions have many moving parts and cannot easily scan objects that do not stay still. Their advantage is the price of the whole system, FreeLSS costing 239 USD (216 EUR) [21], significantly less than Pi3DScan's estimate of 11 000 EUR [2].

3. TECHNICAL IMPLEMENTATION

Goal for the implementation is to create a system of multiple Raspberry Pis, that can be instructed to take pictures simultaneously. User interface is provided through a web browser, to facilitate support for a wide variety of devices. Simultaneous image capture is useful when the target cannot stay still for extended periods of time and multiple different angles of view are required, such as with 3D scanners.

Technical implementation is split to three parts, the multicast servers and clients for service discovery and image capture, web interface and image retrieval. Initially, the server for the web interface was planned to start the Python scripts used for service discovery and image capture, but that turned out to be too complicated and a separate script that runs when the Raspberry Pi is booted was used instead.

Implementation uses Raspberry Pi 3 B with Camera Module v2. Raspbian Buster version July 2019 is used as the operating system as it is the Raspberry Pi Foundation's official operating system and easily available at the Raspberry Pi Foundation's website [22].

3.1. Multicast

IPv4 multicast, with UDP (User Datagram Protocol) sockets to send and receive the messages between Pis was used. All server and client groups use arbitrarily chosen IPv4 addresses and ports within the available addresses and ports. Magic strings, special strings that are unlikely to be sent by other devices, are used as payloads to lessen possible interference from other devices within the network, and to possibly have different Raspberry Pi groups in the same network with different magic strings. Python 3.7 was the programming language of choice.

Python's socket module provides tools for building the UDP sockets. Multicast sockets require some specific socket options. Socket options must be used to allow socket to be immediately reused and to tell the kernel which multicast group the script wants to listen with a membership request.

3.1.1. Service Discovery

Service discovery is a decentralized solution using a pull model, where the Pi unit sends a multicast message asking for other units within the network to answer.

The server script creates a socket that is bound to the multicast group and predetermined port. It then listens for messages constantly in a loop. The client script sends a predetermined magic string to the multicast group. All devices within the network running the server script the answer by sending a different magic string directly to the client. Magic strings have to match exactly between devices or the message is discarded. The client then reads the IP addresses from the answers using Python's `socket.recvfrom()`-method. IP addresses are then appended to a list. After the client has gone through all the messages in its multicast socket's buffer, the list includes the IP addresses of Raspberry Pis or other devices running the service

discovery server, including the client. The client 'discovers' itself the same way it discovers other devices, meaning the client device also has to run the server script.

Client script's communication with devices running the server script happens entirely within a single function. That function is called when the client script is started and when the list of devices in the network needs to be updated. Most of the time the client is listening on a different socket, bound to localhost (127.0.0.1) and arbitrary port. This UDP socket is used when sending the list of device IP addresses to other Python scripts within that same device. It also uses a magic string to identify inquiries. The list has to be serialized before it is sent and deserialized afterwards, which is done with Python's pickle module. If no messages have arrived to the localhost socket after 60 seconds, the socket will timeout. The script catches the timeout error and calls the function to update the list of device IP addresses. After the list has been updated, the script resumes listening with the localhost socket. This loop is continued indefinitely. Activity in the localhost socket resets the 60 second timer, which could lead to scenarios where constant use of some Raspberry Pis features lead to the list being outdated. Different stages of communication between the client and server scripts is visualized in Figure 1.

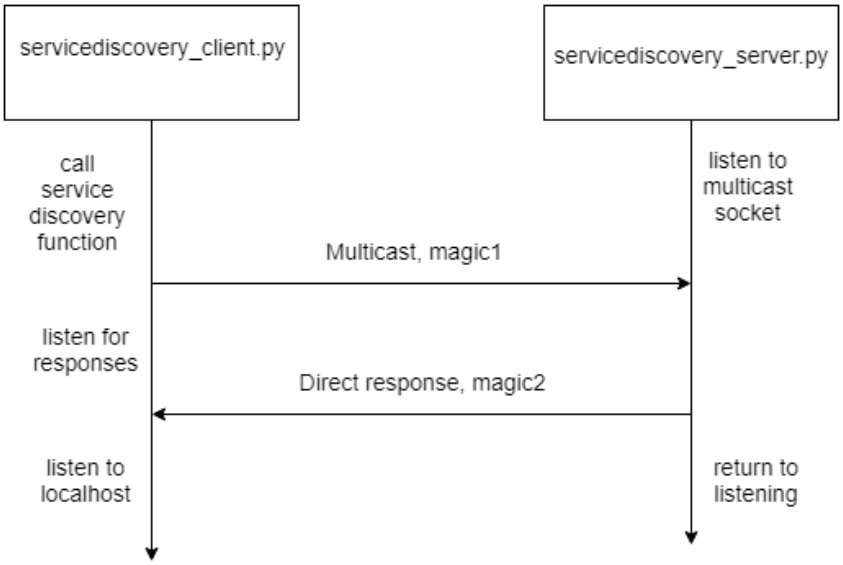

Figure 1. Communication of service discovery client and server

3.1.2. Image Capture

Image capture is triggered when a listen script dedicated to the task receives a predetermined message through multicast. After receiving the message the listen script runs raspistill with Python's subprocess module. Raspistill is one of four camera applications for Raspberry Pi camera. It is meant to be used when capturing still images, and by default captures a single image with the highest supported resolution [23]. The highest supported resolution for Camera Module v2 is 3280x2464, which

is also the resolution used in this implementation. Camera's settings are determined with command line arguments to raspistill, including where the taken image is saved and what it is called. Image's name is the system time when message to capture the image was sent. Images are saved to /var/www/html/ to facilitate web server's access to them.

The image capture message is sent to multicast group from a single Raspberry Pi. The message's payload is a magic string that has to match with the magic string on listen scripts. Having different magic strings on listen scripts in different devices can be used to create multiple 'camera groups' which work independently from each other, despite using the same multicast group and port. Figure 2 visualizes how the user's device only communicates with a single Raspberry Pi camera unit.

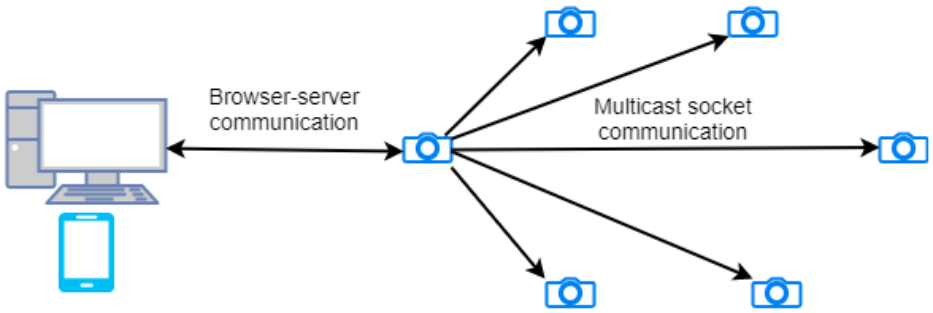

Figure 2. Communication between devices. Router acting as an intermediary is not present in the figure.

3.2. Web Interface

Users interact with Raspberry Pis through a web browser. Apache2 is used as a web server. The interface consists of 3 different web pages, index, images and camera preview stream. Index and images are created using Python scripts, using Common Gateway Interface (CGI). Camera preview is done with Dave Jones' web streaming demo for Pi camera [24], as seen in Figure 3, which runs its own web server at port 8000. CGI web pages work by Python script's being run when the page is asked from the web server. The script's output is then sent to the browser. If the output is correctly formatted HTML, the browser should display it like any other web page.

The index page shown in Figure 4 shows the user the addresses of Raspberry Pis present in the network, provides links to preview stream and images pages and includes the button which triggers image capture. Raspberry Pi's local IP address is fetched by creating a socket and asking for that sockets name, which includes the Pi's IP address. IP addresses of other Pi's in the network is retrieved from service discovery client script, through a localhost socket. The button is standard HTML submit input type and the python script accesses its input through cgi module. The python script waits until button press adds 'Submit1' to `cgi.FieldStorage()`, which triggers image capture proceedings. Image capturing script is then started as a subprocess of the index page.

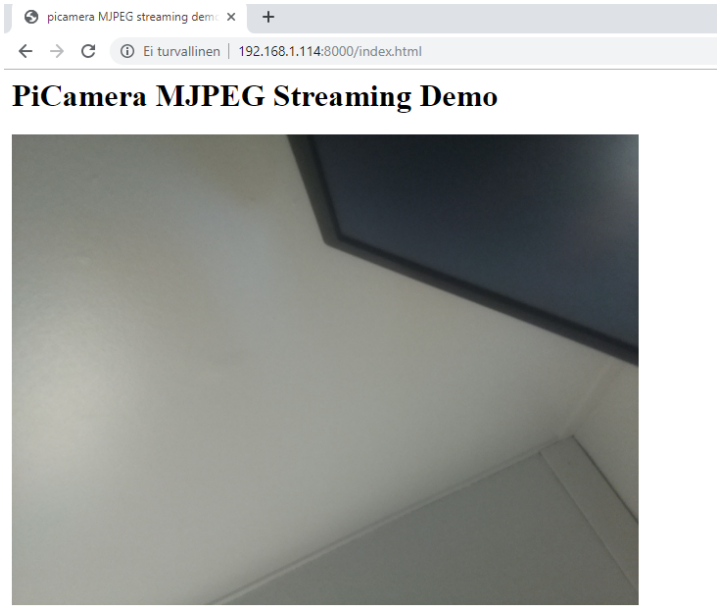

Figure 3. View of Dave Jones' web streaming demo

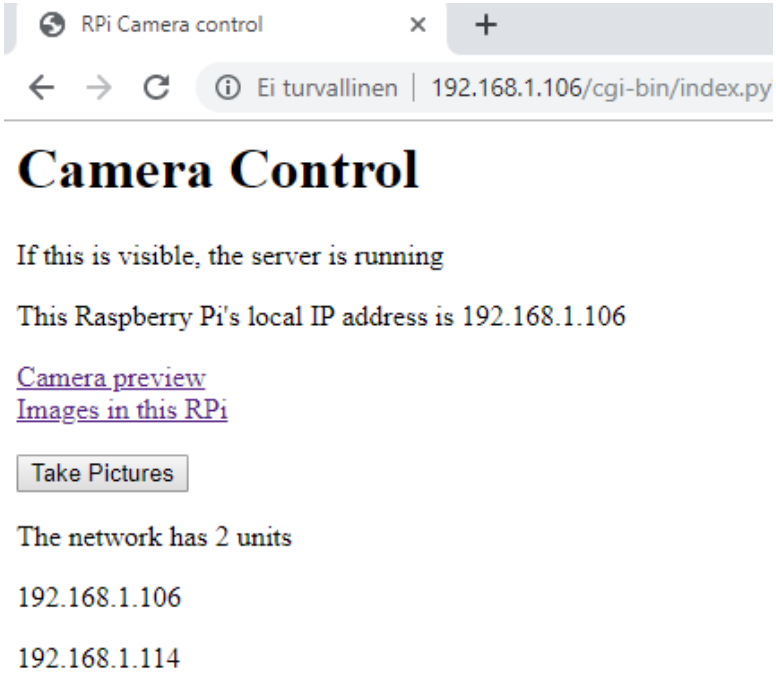

Figure 4. Index.py page

The images page shown in Figure 5 shows the user images and zip-files stored in that Raspberry Pi. User sees a smaller resolution preview of each image and has the ability to download just that image, or to ask the Raspberry Pi to retrieve all the images

with the same name from other Pi units in the network. The resulting zip-file is then downloadable from the page. The page's script iterates through `/var/www/html/` -folder, recognizes images and zip-files, and prints appropriate HTML, including the previews for the images and buttons for creating a zip-file. This can lead to slow loading of the page, depending on the connection speed of user's device and the Raspberry Pi, because all the images are loaded simultaneously. Different images' submit buttons are named differently, and when a submit button is pressed, the Python script checks which image's name is added to form and starts image retrieval and zip-file creation scripts with appropriate arguments. Images and zip-files are shown in arbitrary order, which can lead to difficulties for the user when looking for the correct zip-file after reloading the page.

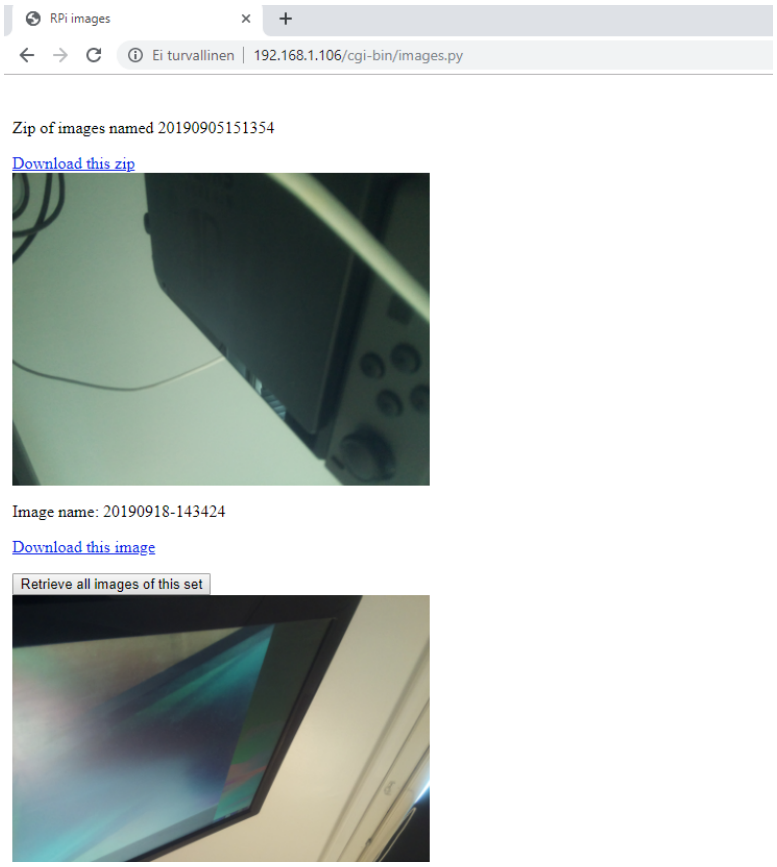

Figure 5. Images.py showing images and zip files stored in the Raspberry Pi

3.3. Image Retrieval

Image retrieval script (named multidload.py) uses urllib.request module to download images and shutil module to create the zip-file. The script takes the name of the images to be retrieved as a command line argument and does not function without it. All images that are retrieved have to share a name for the script to function. Similarly to index page's script, the image retrieval script asks for the addresses of other Raspberry Pi units from the service discovery client script.

The script creates a temporary directory to where the images are downloaded. It then iterates through the list of Raspberry Pis in the network, trying to download an image with a correct name. If that fails, the loop continues with next Raspberry Pis IP address. If an image is downloaded successfully, it is renamed according to its origin IP address. When the script has iterated through the Raspberry Pis, a zip copy of the temporary directory is created and the temporary directory is deleted.

A modified version of the image retrieval can be run on a separate device with the service discovery client script in order to directly download the wanted image set. This can be of use if the user wants to directly download a larger batch of images for processing, as Raspberry Pi's processing power is limited.

3.4. Inter-Script Relations

A separate script is used to start listening scripts when Raspberry Pi is powered on and to restart them if they crash. This is done with subprocess module, using `subprocess.poll()`-method to see if the subprocesses are still running. This script starts image listening, and service discovery client and server scripts. Image listening script then controls the web streaming script, to prevent problems with multiple programs trying to access Raspberry Pi's camera at the same time. Hierarchy of the subprocesses can be seen in Figure 6. It is also possible to run the scripts individually without using the start-up script.

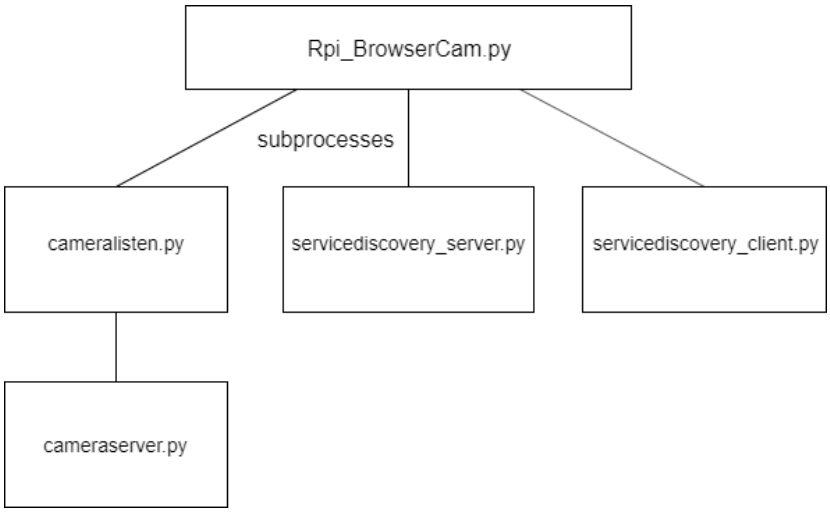

Figure 6. Hierarchy of script subprocesses

Scripts communicate with each other through sockets and some scripts do not handle errors when trying to connect to sockets. Scripts that are supposed to be running indefinitely, such as service discovery server and client, do not require other scripts to be running beforehand, but image retrieval and index page scripts do not run properly if service discovery client script is not already running on that device. Image retrieval script is usually started through index page, so in case service discovery client script is not running, user would get stuck trying to load the index page.

4. RESULTS

Tests concerning concurrent image capture and service discovery are detailed in this chapter. Testing and evaluating these features is important in order to know if the implementation can be used as a basis for a 3D scanner. This includes evaluating synchronization of image capture, and service discovery implementation's scaling for more devices and speed. Other useful tests would be for example testing the speed at which Raspberry Pi can retrieve images from other Raspberry Pis or how different raspistill settings affect image quality. All the tests detailed are limited by the use of only two Raspberry Pis, which means that certain compromises have to be made.

4.1. Concurrent Image Capture

In this work, synchronization of two Raspberry Pis is tested. The purpose of these tests is to see if the Raspberry Pis capture images within a suitable time frame from each other and if wireless local area network (WLAN) is stable enough for synchronized image capture. Concurrent image capture is important when using the cameras for 3D scanning purposes [2]. Certain amount of desynchronization is unavoidable but it should be minimized.

Testing is done with the Raspberry Pis connected to Sagemcom F-3686AC cable modem via WLAN connection. During the test, the Raspberry Pis takes photos of a screen which shows a console view with prints of an increasing counter. Prints happen at roughly every 22 milliseconds (measured with the python script itself) and the 60Hz screen updates every 16.66 milliseconds, which means that the screen should update between every print.

Raspberry Pi camera module is a rolling shutter camera, which means that all pixels in the image are not captured at the exact same moment. This leads to some distortions when capturing frequently changing scenes, such as a 60Hz computer screen. In certain test cases the number on the screen is ambiguous. This and the limits of the screen lead to a certain amount of measurement error when analysing the rest results but it is good enough to see if the target of the image capture has time to change drastically between the two images. Pi3DScan's 3D scanner takes two sets of images with 200 milliseconds between the image sets [2] and both image sets are used to create a single scan. In the worst case scenario the measurement error is just over 60 milliseconds, assuming the worst possible timings with the screen updating (16.66 milliseconds) and prints happening (22 milliseconds), and assuming ambiguous text on the captured image leading to misreading of the latest print (22 milliseconds again). Because of these factors six prints or roughly 132 milliseconds is used as the highest acceptable desynchronization between the two images.

The test is repeated ten times. Occurrence of each print delta between the captured images is visualized in Figure 7.

The maximum desynchronization of 6 prints happened once during the testing. Average print delta was 3.5, median 4. Concurrent image capture is successful with WLAN and it is synchronized enough for 3D scanner use. Test results might however change if more devices are added to the network or with different WLAN set-ups.

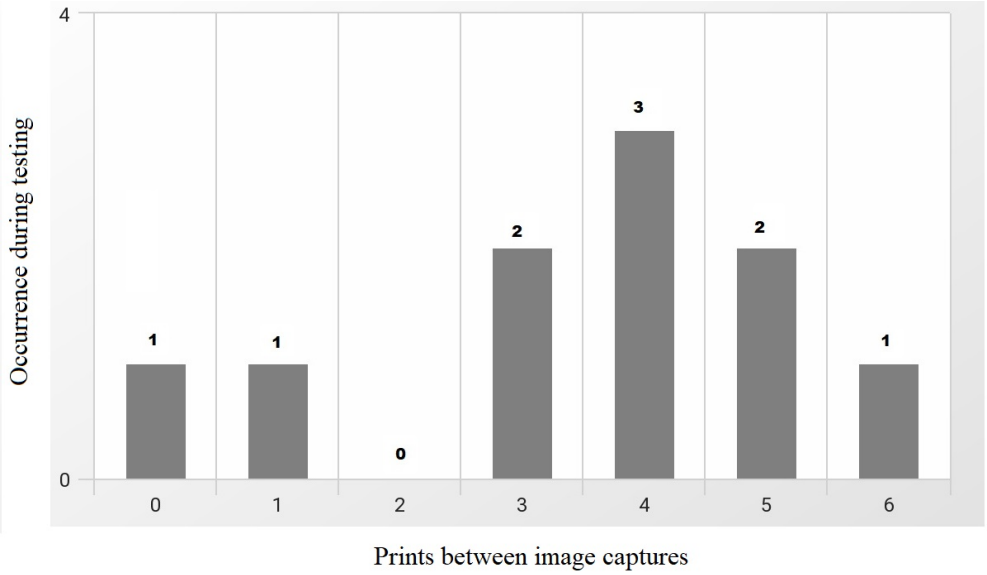

Figure 7. Occurrence of the delta between images

4.2. Service Discovery Device Limits

Maximum amount of devices found with the service discovery implementation is tested. The purpose is to see how many devices could answer a Raspberry Pi's service discovery client multicast, without socket buffers being filled or other limits being reached. Support for a large amount of devices is necessary for many IoT solutions and 3D scanners, such as Pi3DScan's 3D scanner that uses 100 cameras [2].

Test is done by having a single Raspberry Pi answer another Raspberry Pi's service discovery client several times. The implementation treats each answer as a new device and thus, it does not matter if the answers come from 100 unique devices or one device 100 times. The limiting factors should be the buffer of the service discovery client during listening for responses, size of the list sent through the local socket and the ability of the index page to show a list of the devices.

The test is done with 10, 100, 250 and 500 answers per service discovery search. Tests with 10, 100 and 250 answers work without issues but with 500 answers internal server error appears when trying to load the index page. This is because of a bottleneck with the internal socket that is used to send the list of devices to index page script. Changing `socket.recvfrom()`-method's buffer from 10240 to 20480 when retrieving the pickled list from the internal socket fixes the error and afterwards the service discovery works normally even with 500 answers.

The implementation's ability to process hundreds of devices found with service discovery is adequate. A list as large as 500 device IPs is possible to send through the sockets and display for the user.

4.3. Service Discovery Response Time

The time it takes for two Raspberry Pis to respond to a service discovery client's multicast message and for the answers to be processed is measured. This test is done to see if the Raspberry Pis answer to service discovery requests at a reasonable delay.

Service discovery client runs on a Windows 10 PC. The PC and the Raspberry Pis are connected to Sagemcom F-3686AC cable modem via WLAN connection. Time is measured with Python's time-module, starting before the initial message is sent and stopped when the socket is closed after socket timeout. Socket's timeout time was set to one second, which is afterwards subtracted from the measured times. Test is run ten times. Results can be seen in Figure 8.

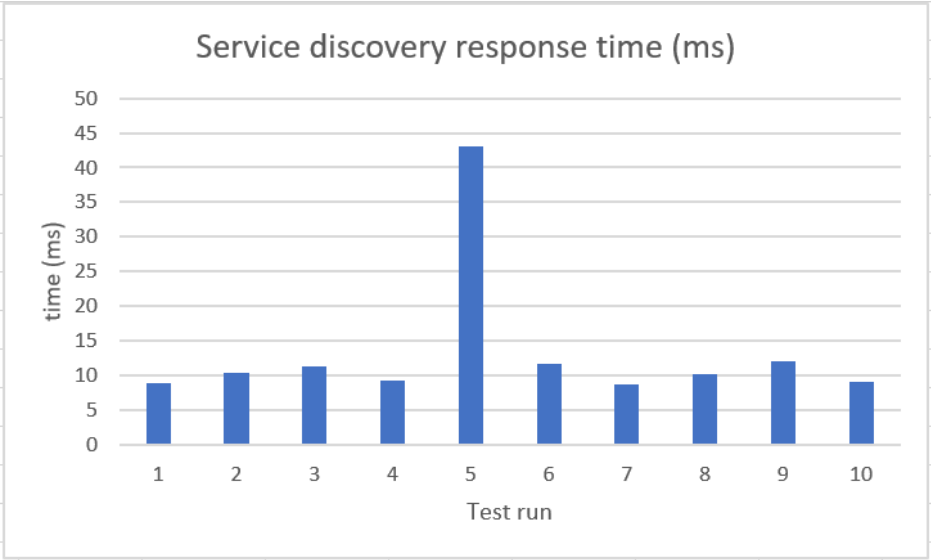

Figure 8. Response time measured with two Raspberry Pis

Average response time of the ten tests is 13.4 milliseconds, median 10.2 milliseconds. Nine of the ten test runs have only small differences between them, all are around 10 milliseconds. The fifth test has the worst response time 43.0 milliseconds. Test results indicate that the service discovery implementation works fast even in the worst test and that the socket timeout time could probably be reduced from 1 second without any issues. Adding more Raspberry Pis should not affect the needed socket timeout, but would increase the overall time service discovery client spends processing the answers. Changing the network setup could lead to different the test results.

5. DISCUSSION

In this project development of a bare-bones concurrent multi camera system for Raspberry Pi is detailed. The implementation achieves the goal of concurrent image capture, even when using wireless connections. System's Raspberry Pis can function independently or together and the user can use the system's browser based interface with different devices. Developed service discovery is sufficient according to tests done. System's Raspberry Pis connect to each other automatically and a designated script makes sure all the needed scripts are running. The system's user can download images taken by different Raspberry Pis to their device of choice.

When compared to Pi3DScan's [2] or Straub and Kerlin's [3] 3D scanners the system presented in this paper is less reliant on a single master device such as a Windows computer controlling the Raspberry Pis, but it is also lacking in some features.

The system gives its user the ability to download images taken by different Raspberry Pis, but all the images have to be zipped by one Raspberry Pi before downloading is possible. This creates a bottleneck for how fast images can be downloaded because each image has to be downloaded twice. Once when the Raspberry Pi collecting the images retrieves it from another Raspberry Pi, and the second time when the user downloads the zipped images. This is inefficient use of Raspberry Pis' processing power and network's bandwidth and creates an extra step for the user downloading the images. Pi3DScan's 3D scanner [2] downloads all the images from different Raspberry Pis using their own software running on a Windows computer. For Straub and Kerlin's [3] 3D scanner their Raspberry Pis send images to a file server, where their user process will then retrieve them. It would be possible to slightly modify the image retrieval script running on the Raspberry Pis in order to directly retrieve the images to a computer or even to build an interface around that script, coupled with the service discovery client to learn IP addresses of Raspberry Pis.

Currently the system does not give users ability to delete images from Raspberry Pis memory cards. This means that eventually the SD cards will not have enough space for more images and the system will cease to function. Deleting feature could be implemented to the images page.

The system does not give the user ability to capture images on the Raspberry Pis separately. Only way to capture images on just a single Raspberry Pi is to have it use different magic strings than other Raspberry Pis in the network. Changing those magic strings is not possible through the user interface, the user would have to change them directly on the Python scripts. If the user would wish to for example test functioning of individual Raspberry Pi units, it would probably be easier to just turn all other Raspberry Pis off. This also severely limits how the system could be used for other purposes than 3D scanners, such as surveillance cameras. An option to only capture images on one specific Raspberry Pi could be implemented to the user interface. It could function by starting a modified version of image capturing script that only sends the message locally to the listening script on that Raspberry Pi.

The system's user interface has no way for changing settings the Raspberry Pis use for capturing images. Pi3DScan's open source scripts [25] allow the user to change raspistill options, FreeLSS [19] gives its user option to choose scan detail and laser settings, and FabScanPi's [20] scanner includes the possibility for the user to change several different settings regarding how the scan is performed. Changing raspistill

settings for the system implemented required changing them inside the Python scripts, which can take a lot of time if many Raspberry Pis are used. Raspistill options could be implemented to the user interface e.g. users could input their wanted settings as text commands through the web interface and those settings would then be integrated into multicast message, similarly to Pi3DScan's open source scripts [25].

Service discovery solution developed does not carry information about what kind of devices are discovering each other. This is not a problem when all Raspberry Pis in the network are identical but there are several different Raspberry Pi models and cameras which have different capabilities. This information could be packaged to the service discovery responses and visualized to the user in some way. Older and less powerful Raspberry Pi models could have difficulties running both the necessary scripts for image capture and the web server at the same time but could still be useful as basic cameras. Repurposing the service discovery implementation to a system where some devices have fundamentally different purposes, such as adding a file server for the images as is done in Straub and Kerlin's scanner [3] would require adding at least basic identification of device's features in the service discovery answer.

Website could be rebuilt with JavaScript instead of pure Python and HTML. This would allow the website to be dynamic instead of requiring reloads between every action and provide feedback about how the image capturing is proceeding for the users to see. Fabscan Pi [20] even shows the actual scan data for the user. Rebuilding the website would also give a chance to focus more on the user experience and usability of the interface in order to encourage usage.

Support for lighting control through Raspberry Pis' GPIO pins could be implemented similarly to Pi3DScan's scanner. Having the Raspberry Pis control the lighting of the scanning area is useful for making sure all scans are performed in good conditions. The same script that starts raspistill could also control the lighting or toggle for the lighting could be implemented to the website interface. Lighting could also be used to give user feedback on when the scan itself happens, considering there is a reasonable delay after the user gives the command to start the scan.

6. SUMMARY

In this work a development of a concurrent image capture system for Raspberry Pis is detailed. The system is designed so that it could be used as a basis for a 3D scanner. The final system uses a service discovery solution to facilitate Raspberry Pis working together as a unit. User interface is provided through a website with the server running on each Raspberry Pi.

Evaluation of image capture synchronization and service discovery implementation is performed. The results show the implementation achieves concurrent image capture and service discovery can be scaled to enough devices. Implementation's features are contrasted with other 3D scanners. Implementation's function as a 3D scanner is not tested.

7. REFERENCES

[1] Straub J., Kading B., Mohammad A. & Kerlin S. (2015) Characterization of a large, low-cost 3d scanner. Technologies 3, pp. 19–36.

[2] PI3DScan info. http://www.pi3dscan.com/index.php/instructions. Accessed: 2019-10-20.

[3] Straub J. & Kerlin S. (2014) Development of a large, low-cost, instant 3d scanner. Technologies 2, pp. 76–95. URL: https://www.mdpi.com/2227-7080/2/2/76.

[4] Raspberry Pi frequently asked questions. https://www.raspberrypi.org/documentation/faqs/. Accessed: 2019-07-30.

[5] Vujović V. & Maksimović M. (2014) Raspberry pi as a wireless sensor node: Performances and constraints. In: 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1013–1018.

[6] Patchava V., Kandala H.B. & Babu P.R. (2015) A smart home automation technique with raspberry pi using iot. In: 2015 International Conference on Smart Sensors and Systems (IC-SSS), pp. 1–4.

[7] Lewis A., Campbell M. & Stavroulakis P. (2016) Performance evaluation of a cheap, open source, digital environmental monitor based on the raspberry pi. Measurement: Journal of the International Measurement Confederation 87, pp. 228–235. Cited By 21.

[8] Vujović V. & Maksimović M. (2015) Raspberry pi as a sensor web node for home automation. Computers and Electrical Engineering 44, pp. 153–171. Cited By :91.

[9] Al-Fuqaha A., Guizani M., Mohammadi M., Aledhari M. & Ayyash M. (2015) Internet of things: A survey on enabling technologies, protocols, and applications. IEEE Communications Surveys and Tutorials 17, pp. 2347–2376. Cited By 1822.

[10] Munir S.A., Xie Dongliang, Chen Canfeng & Jian Ma (2009) Service discovery in wireless sensor networks: Protocols classifications. In: 2009 11th International Conference on Advanced Communication Technology, vol. 02, vol. 02, pp. 1007–1011.

[11] Quevedo J., Antunes M., Corujo D., Gomes D. & Aguiar R. (2016) On the application of contextual iot service discovery in information centric networks. Computer Communications 89-90, pp. 117–127. Cited By 16.

[12] Cirani S., Davoli L., Ferrari G., Léone R., Medagliani P., Picone M. & Veltri L. (2014) A scalable and self-configuring architecture for service discovery in the internet of things. IEEE Internet of Things Journal 1, pp. 508–521.

[13] Sruthy S. & George S.N. (2017) Wifi enabled home security surveillance system using raspberry pi and iot module. In: 2017 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems (SPICES), pp. 1–6.

[14] Stojanoski H., Bogatinoska D.C., Salem A.M. & Srebrenkoska V. (2017) Practical, cheap smart home implementation with general purpose embedded hardware raspberry pi. In: 2017 Eighth International Conference on Intelligent Computing and Information Systems (ICICIS), pp. 335–341.

[15] Rasyid M.U.H.A., Saputra F.A. & Prasetiyo A. (2018) I-on smart controller: Portable smart home solution based on arduino and raspberry pi. In: 2018 International Conference on Applied Science and Technology (iCAST), pp. 161–164.

[16] Quadri S.A.I. & Sathish P. (2017) Iot based home automation and surveillance system. In: 2017 International Conference on Intelligent Computing and Control Systems (ICICCS), pp. 861–866.

[17] Huu-Quoc Nguyen, Ton Thi Kim Loan, Bui Dinh Mao & Eui-Nam Huh (2015) Low cost real-time system monitoring using raspberry pi. In: 2015 Seventh International Conference on Ubiquitous and Future Networks, pp. 857–859.

[18] Machidon O.M. & Olaru G. (2015) A service-oriented fpga-based 3d model acquisition system. Advances in Electrical and Computer Engineering 15, pp. 101–107.

[19] FreeLSS website. `http://www.freelss.org/`. Accessed: 2019-10-20.

[20] FabScanPi website. `https://fabscan.org`. Accessed: 2019-10-20.

[21] Murobo store. `http://store.murobo.com/atlas-3d-kit/`. Accessed: 2019-08-21.

[22] Raspbian download. `https://www.raspberrypi.org/downloads/raspbian/`. Accessed: 2019-08-30.

[23] Raspicam documentation. `https://www.raspberrypi.org/documentation/raspbian/applications/camera.md`. Accessed: 2019-09-13.

[24] Dave Jones' streaming demo. `https://picamera.readthedocs.io/en/latest/recipes2.html#web-streaming`. Accessed: 2019-09-13.

[25] PI3DScan open source scripts. `http://www.pi3dscan.com/index.php/download/item/basic-python-scripts-2`. Accessed: 2019-10-20.